**FIG. 1**

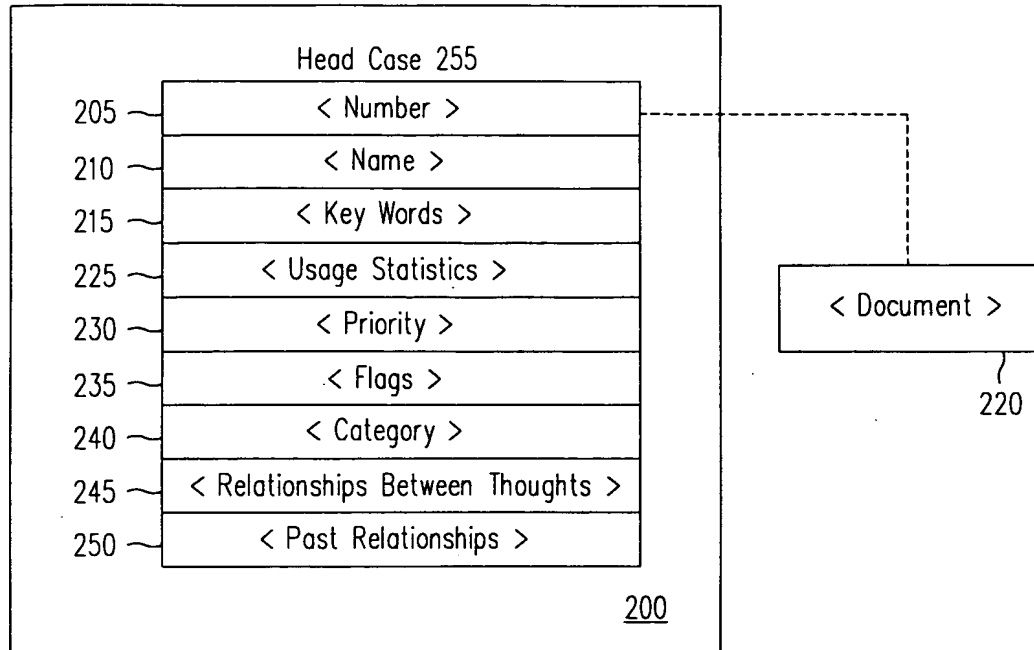


FIG. 2

Diagram illustrating a 'Properties' dialog box (labeled 710) for a 'Cortex' entity. The dialog box contains the following fields and controls:

- Entity Name:** Cortex
- Key Words:** software brain metaphors thought innovative
- Category:** Company (with a dropdown arrow) and a button labeled 'Categories'
- Time Information:**
 - Created: May 30, 96, 09:57:13 PM
 - Modified: May 30, 96, 09:57:13 PM
 - Total Time: 0 days, 01.06.59
 - History button
- OK button**

FIG. 7

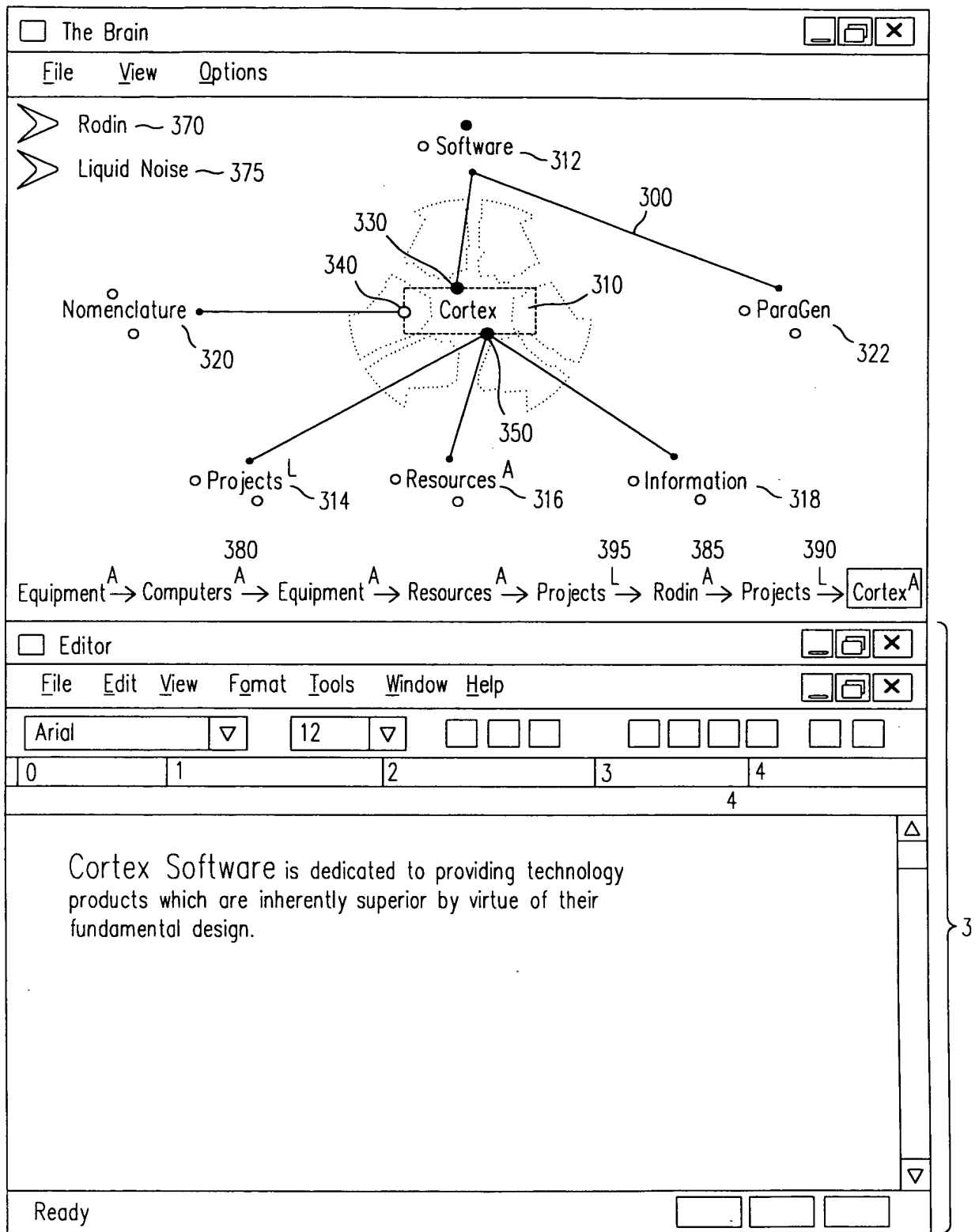


FIG. 3

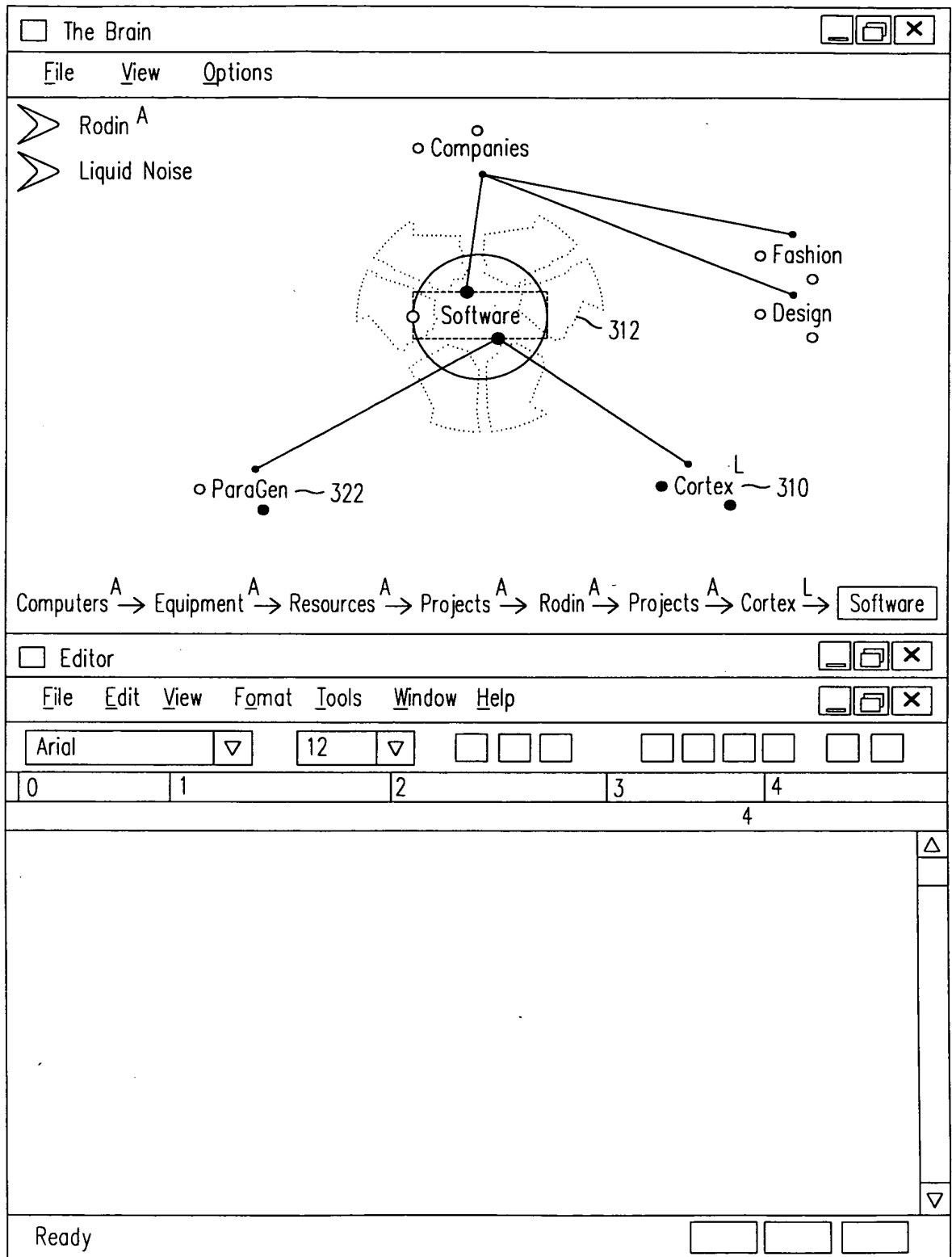
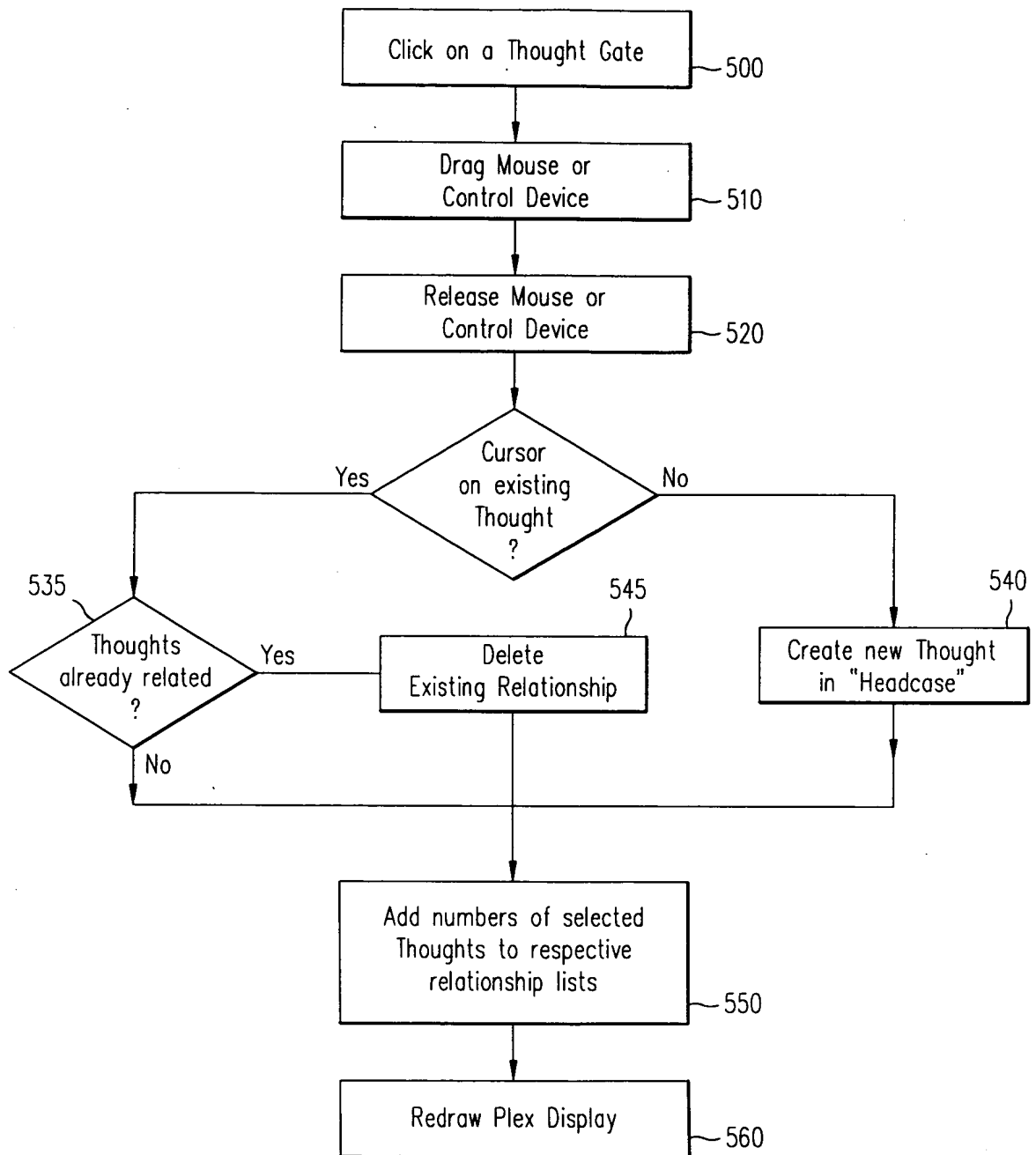


FIG. 4

**FIG. 5**

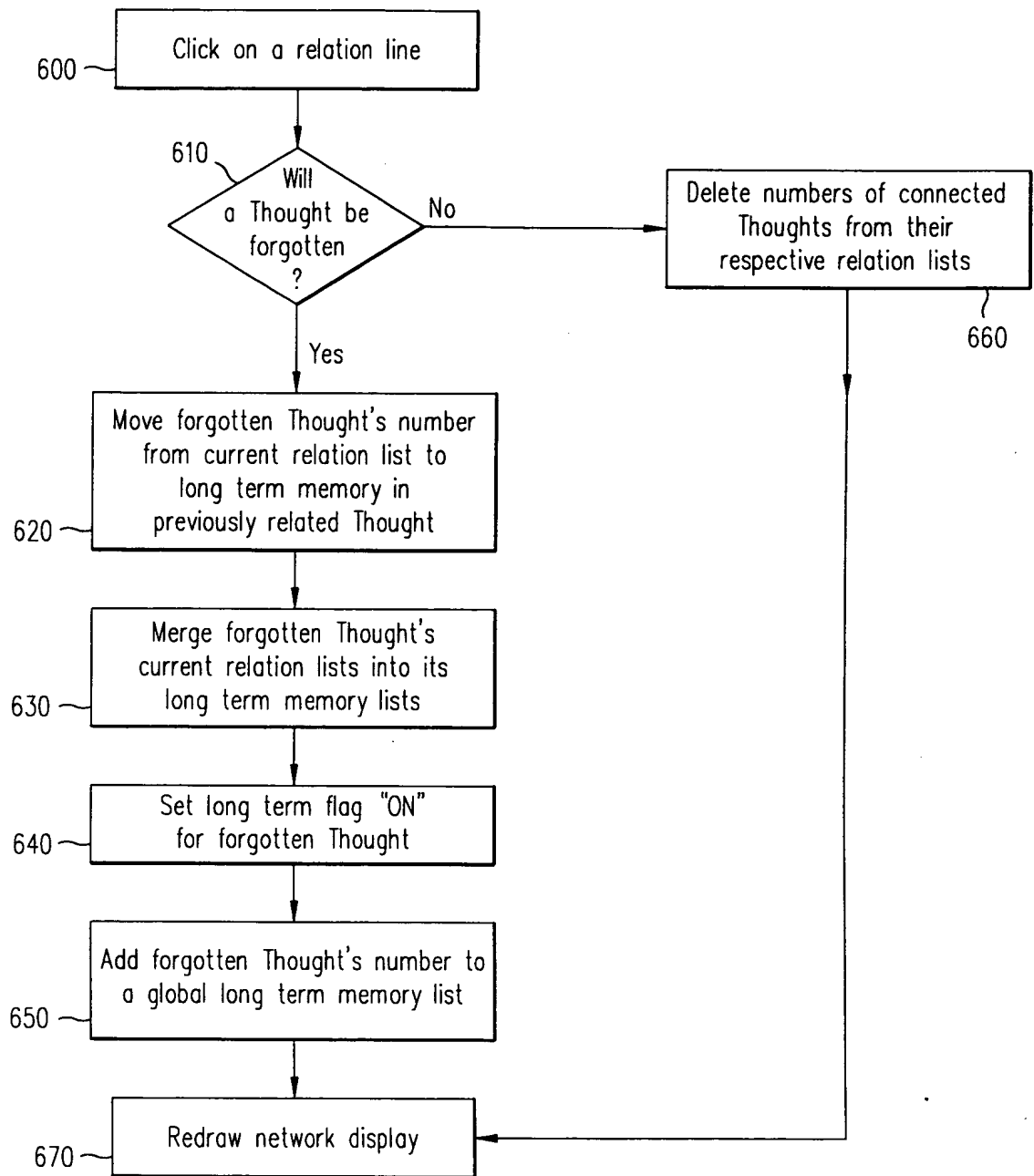


FIG. 6

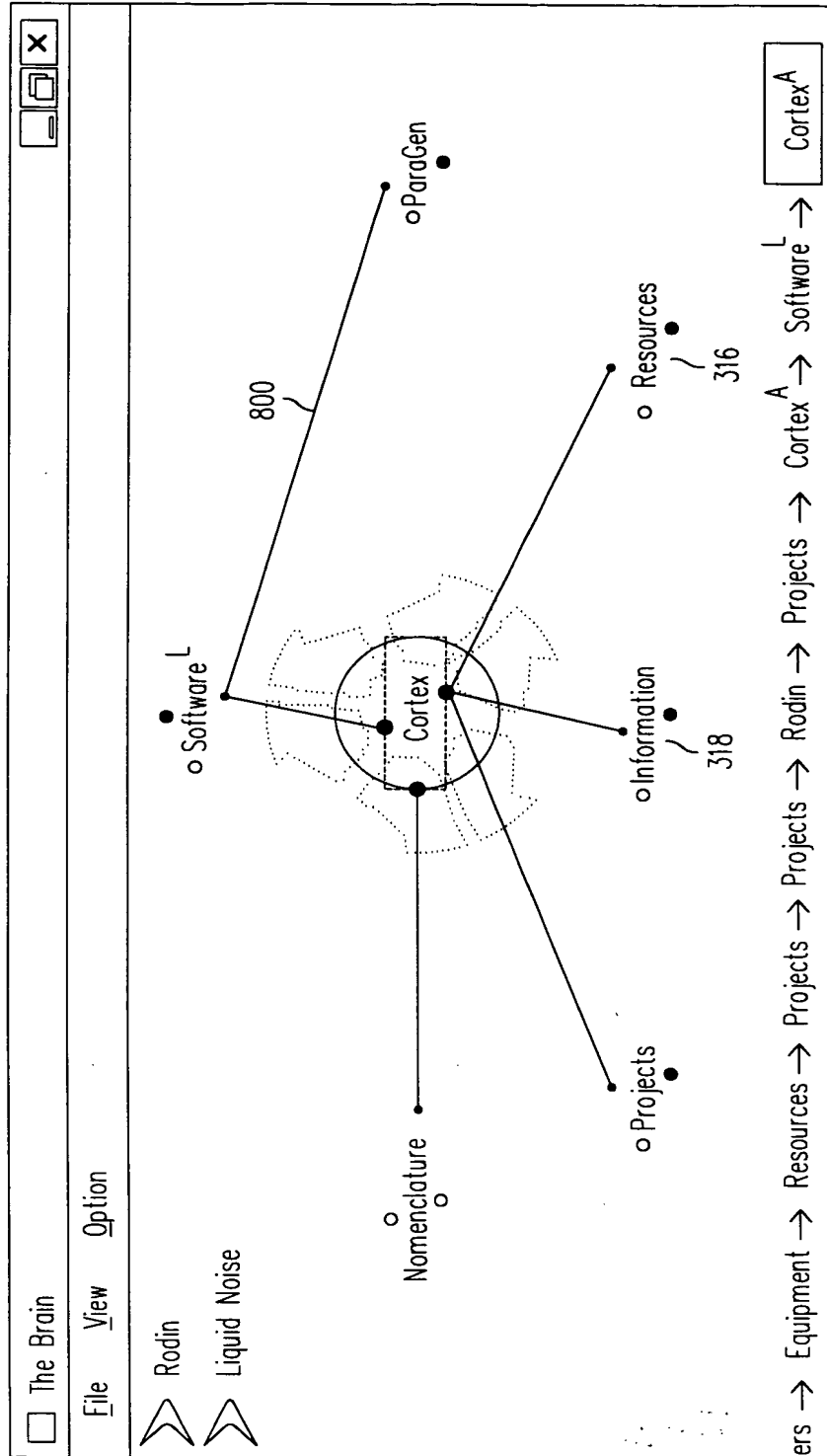


FIG. 8

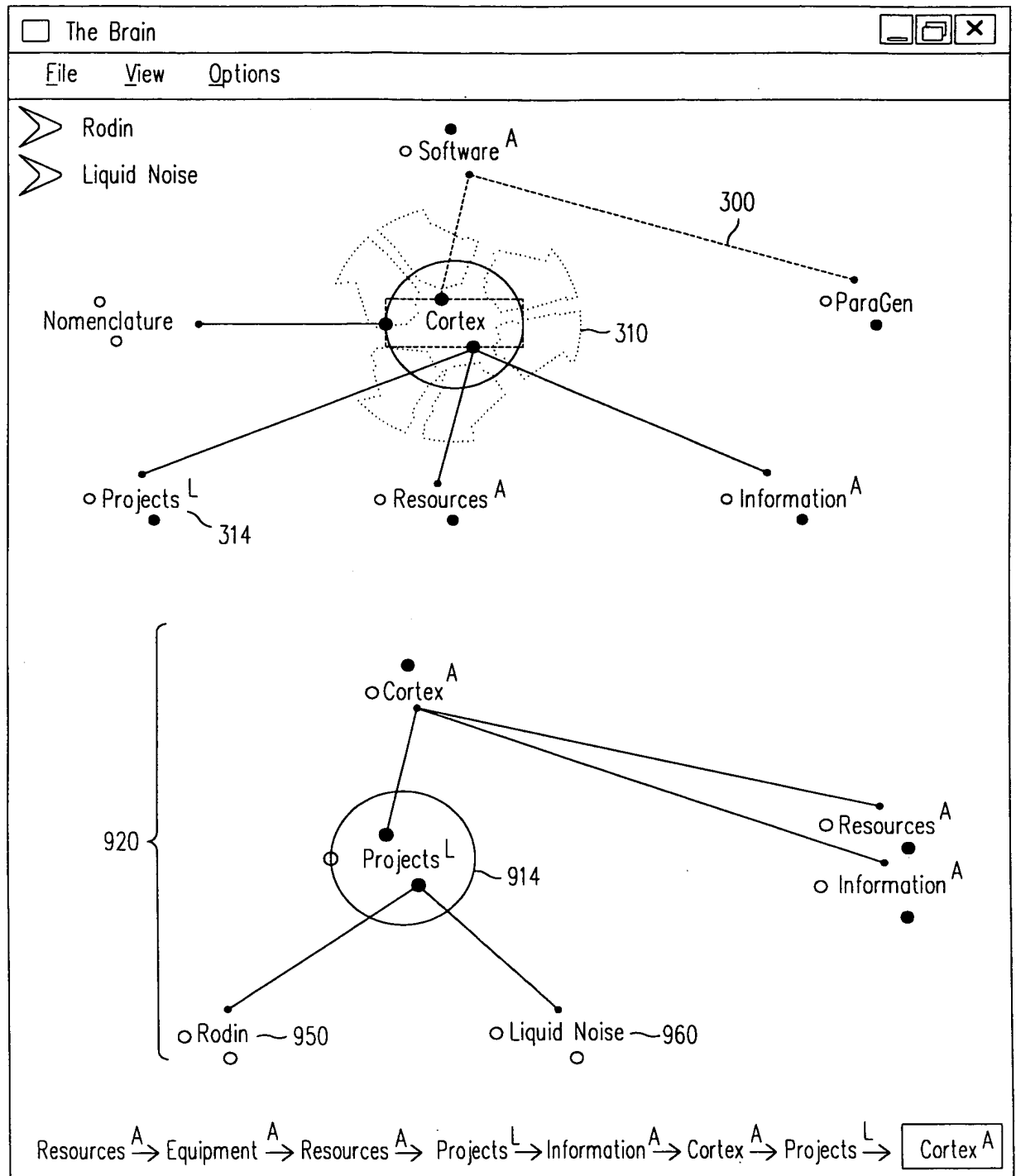


FIG. 9


```

boolean CheckForIsolation(int centralThought, int targetThought)
{
    // this function checks if centralThought is related to targetThought
    // via any of targetThought's relations (not directly)

    // remove centralThought as a direct relation from targetThought
    RemoveRelation(targetThought, centralThought);

    // create an empty thought list to keep track of the search
    intList searchList=CreateEmptyList();

    // start recursive searches on each of targetThought's direct relations
    int relation=GetFirstRelation(targetThought);
    boolean found;
    do {
        found=Search(relation, centralThought, searchList);
        if(found) {
            // centralThought was found, no need to search any further
            break;
        }
        // this loop will end when there are no more relations
    } while(relation=GetNextRelation(targetThought));

    // add centralThought back onto targetThought as a relation
    AddRelation(targetThought, centralThought);

    return found;
}

```

FIG. 10a

Figure 10, cont'd

```

boolean Search(source, dest, searchList)
{
    if(Find(source, searchList)) {
        //source has already been searched
        return FALSE;
    }

    //add source to the searchList
    Add(source, searchList)

    if(source == dest) {
        //this is the destination, we have found it
        return TRUE;
    }

    //recursive searches on each of sources direct relations
    int relation=GetFirstRelation(source);
    boolean found;
    do {
        found=Search(relation, dest, searchList);
        if(found) {
            //centralThought was found, no need to search any further
            break;
        }

        //this loop will end when there are no more relations
    } while(relation=GetNextRelation(targetThought));

    return found;
}

```

FIG. 10b

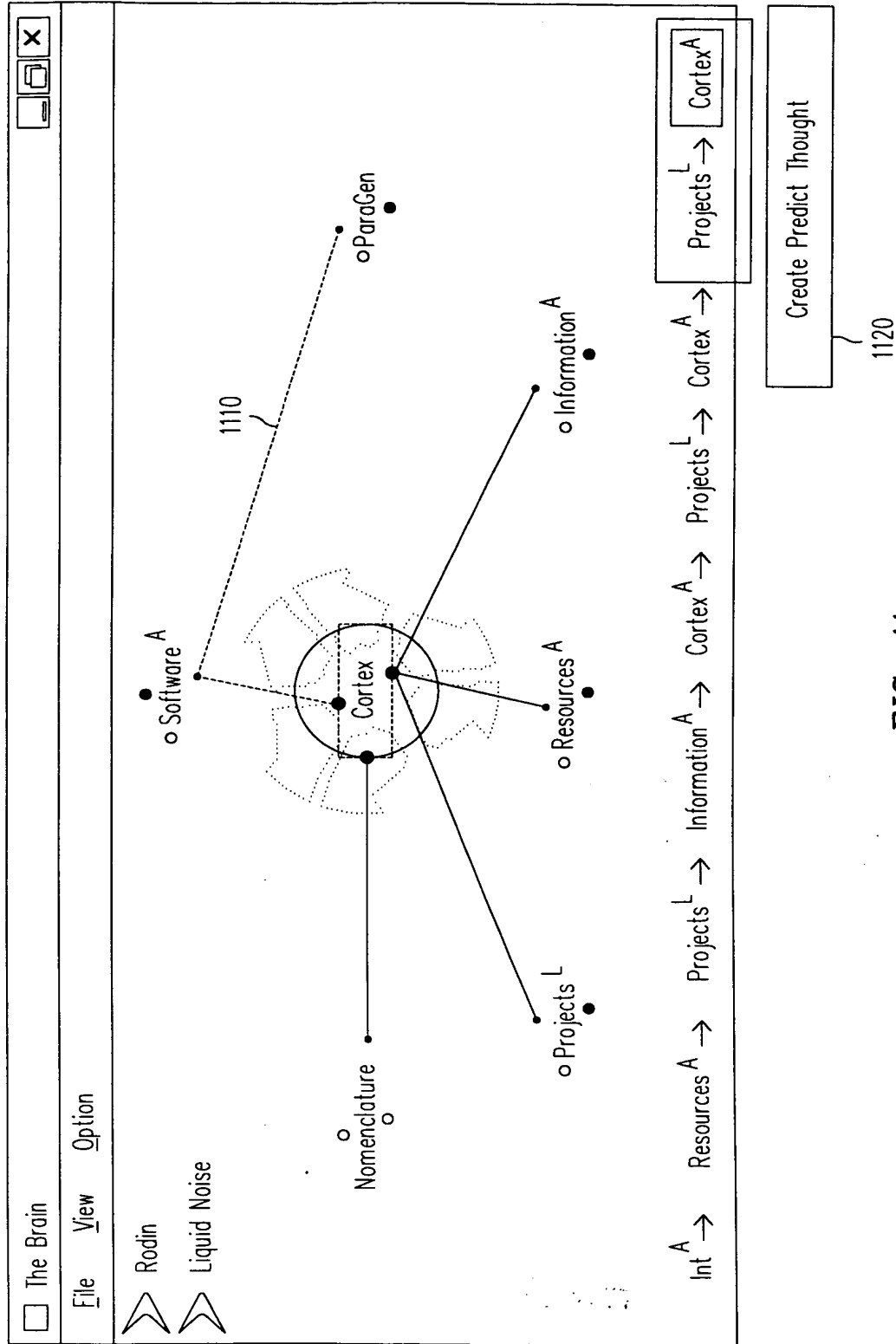


FIG. 11

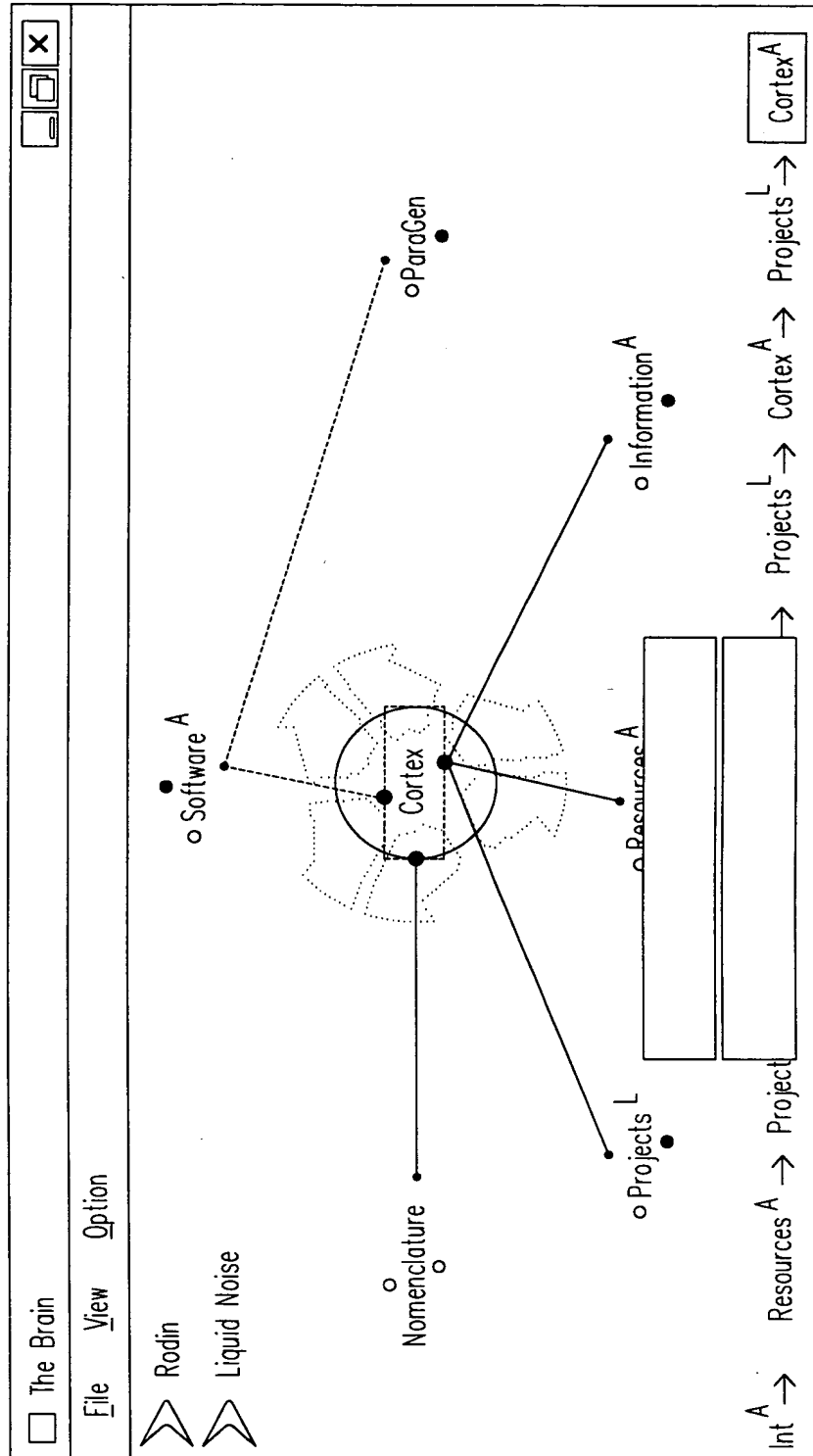


FIG. 12

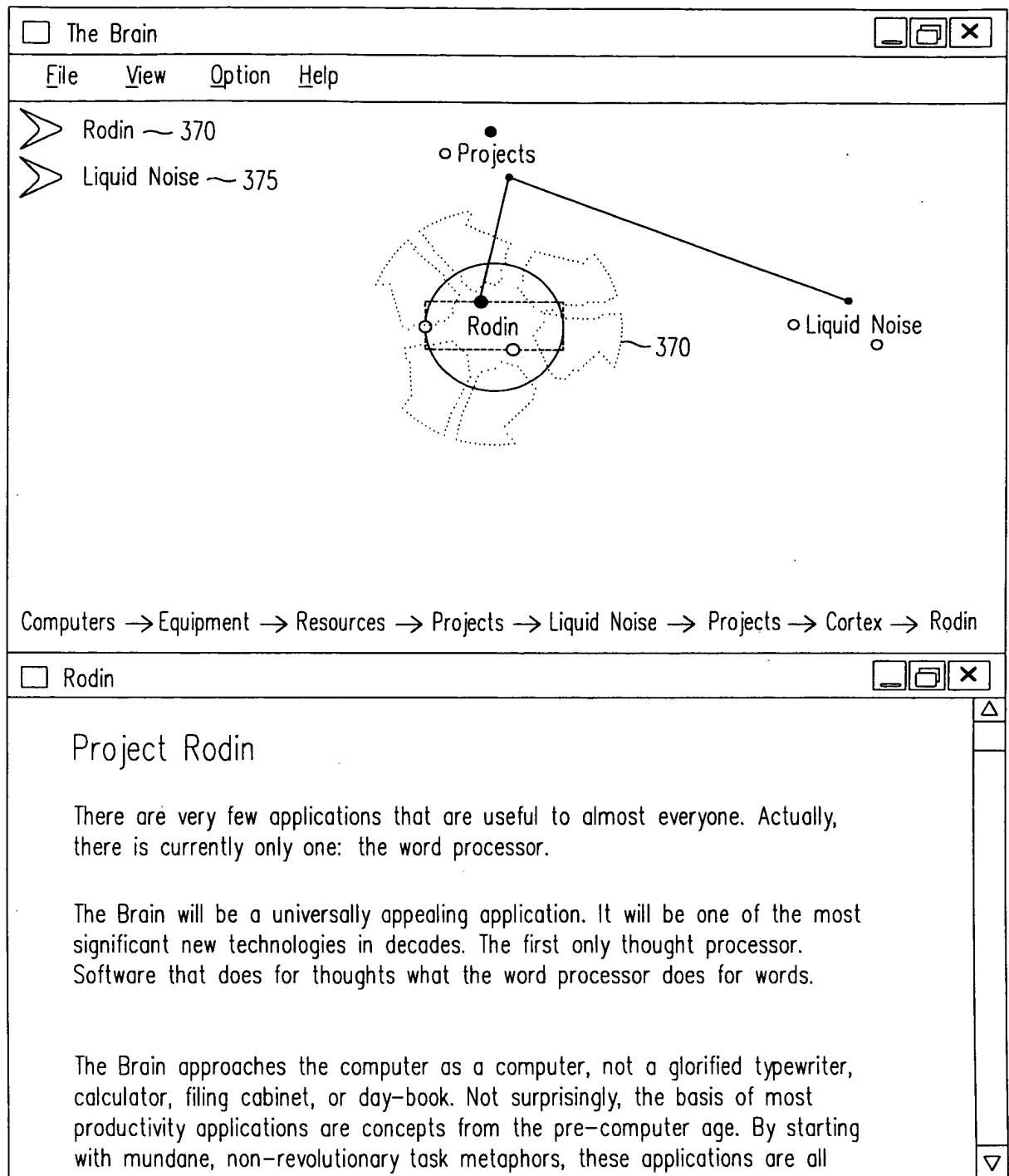


FIG. 13

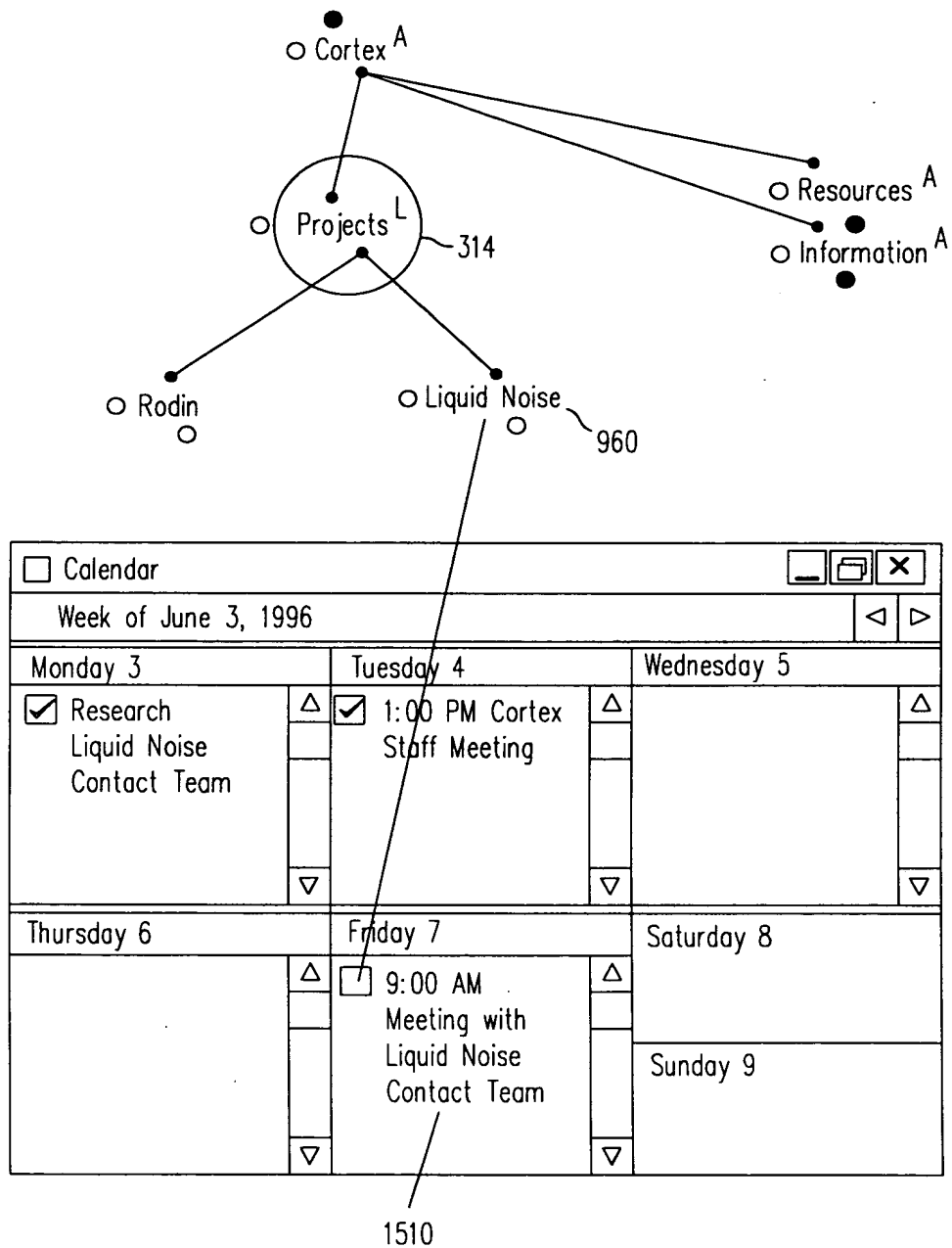


FIG. 15

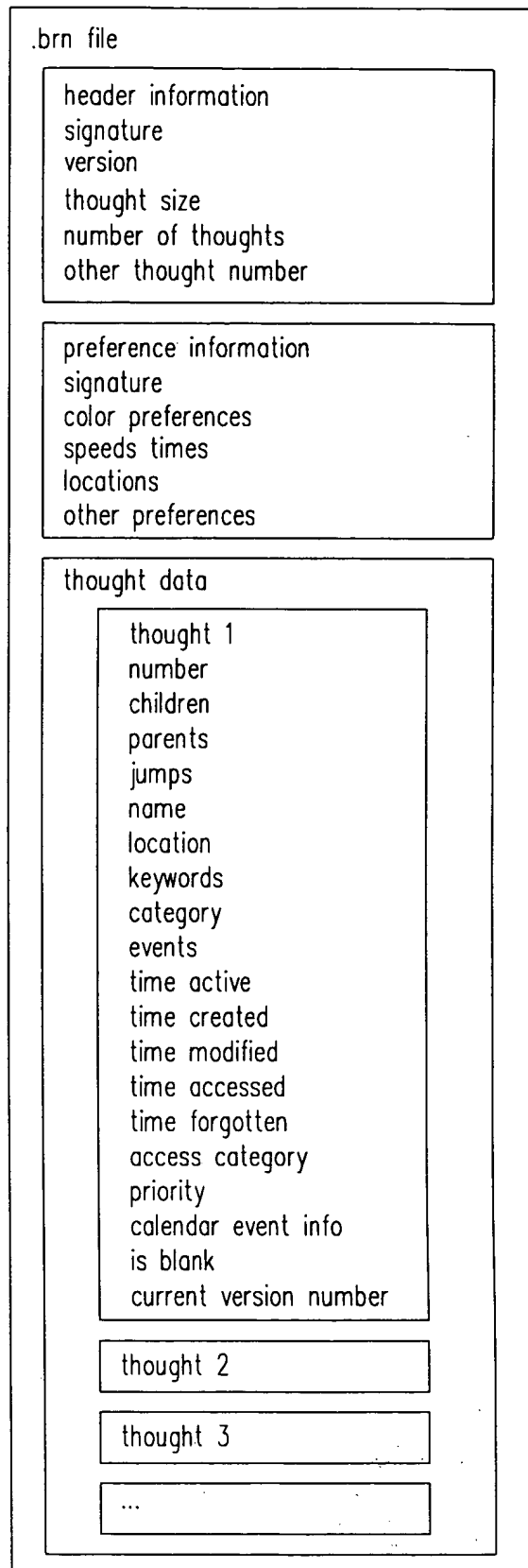


FIG. 16

ForgetThought (fNum)

```

{
    //mark all the children of the selected thought
    list.Clear();
    MarkChildren (fNum, list);
    //unmark the active thought
    list.RemoveThought (activeThought);
    //unmark thoughts with unmarked parents
    INum = list.GetFirstNum();
    while(INum != 0)
    {
        if(INum != fNum)    //don't unmark the selected thought
        {
            pNum = GetFirstThoughtParent (INum);
            while(pNum != 0)
            {
                if(list.Contains(pNum) == FALSE)
                {
                    if(IsThoughtInLongTermMemory (pNum) == FALSE)
                    {
                        //unmark all the children of the unmarked parent
                        childList.Clear();

                        MarkChildren(pNum, childList);
                        list.RemoveList(childList);
                    }
                }
                pNum = GetNextThoughtParent(INum);
            }
        }
        INum = list.GetNextNum();
    }

    //now forget all the thoughts left on the list
    INum = list.GetFirstNum();
    while(INum != 0)
    {
        ForgetThought(INum);
        INum = list.GetNextNum();
    }
}

RememberThought(rNum)
{
    //mark all the children of the selected thought
    list.Clear();

```

```

    MarkChildren(rNum, list);
    // remember all the thoughts on the list
    INum = list.GetFirstNum();
    while(INum != 0)
    {
        RememberThought(INum);
        INum = list.GetNextNum();
    }
}

MarkChildren(num, list)
{
    list.AddThought(num);
    cNum = GetFirstChild(num);
    while(cNum != 0)
    {
        MarkChildren(cNum, list);
        cNum = GetNextChild(num);
    }
}

```

FIG. 17

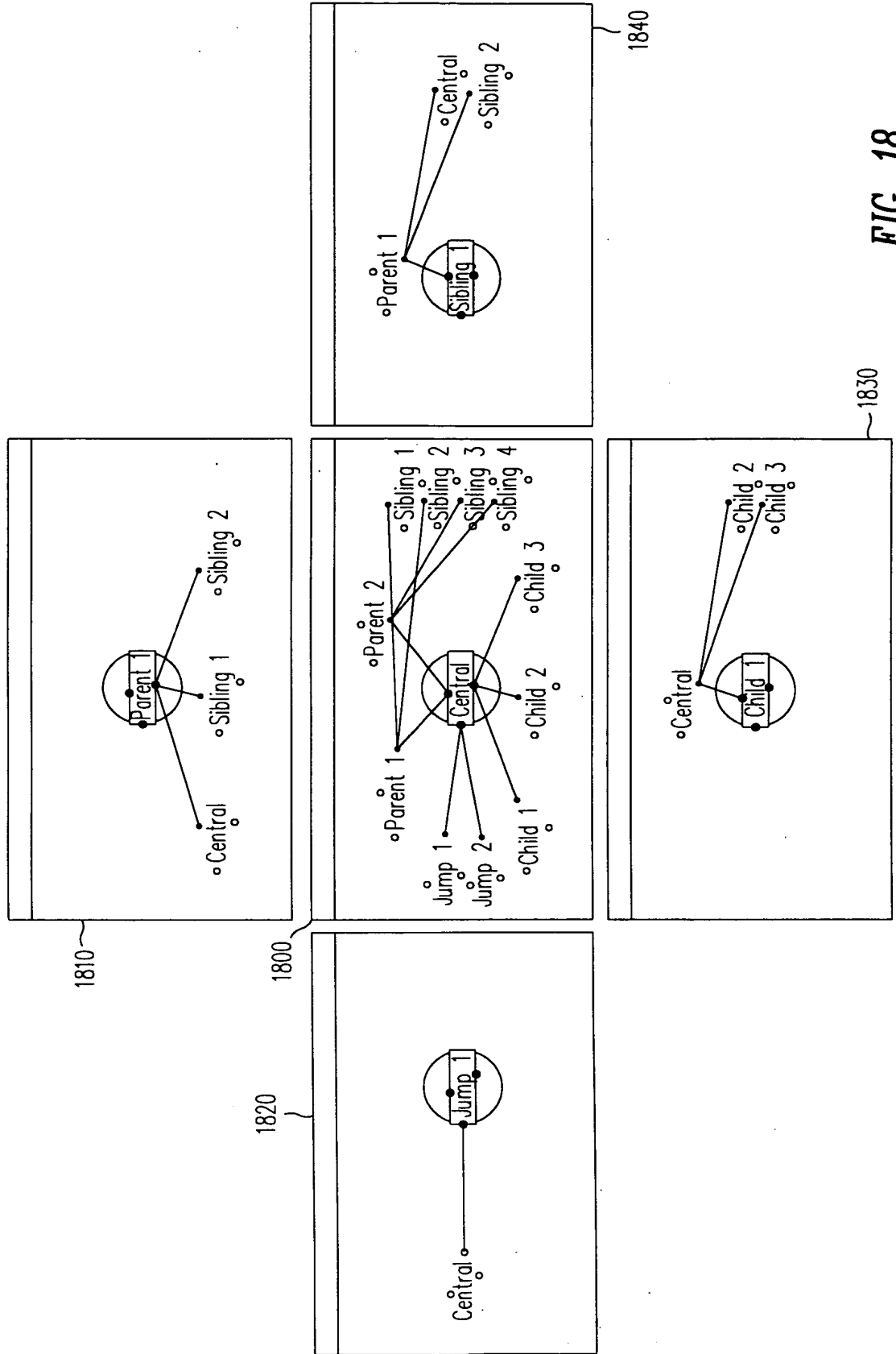


FIG. 18

Database

Cortex

Key Words software brain metaphors thought innovative

Categories Company Categories

Address 9701 West Pico Blvd. #205

City Los Angeles

State CA

ZIP 90035

Telephone 310-552-2541

Fax 310-552-2841

e-mail cortex@cinenet.net

FIG. 14

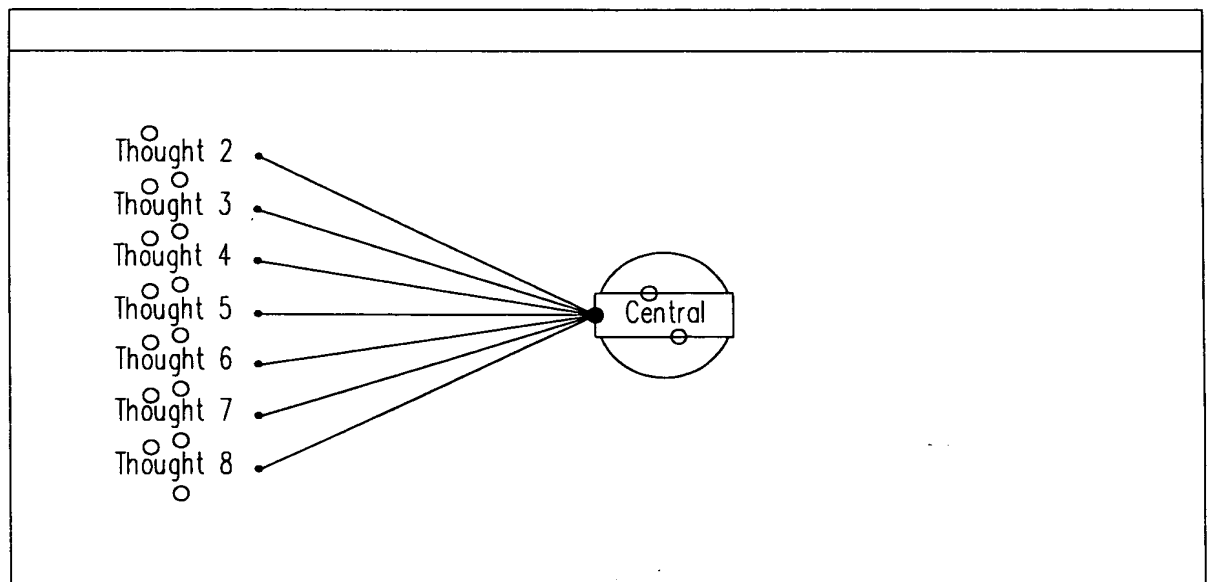
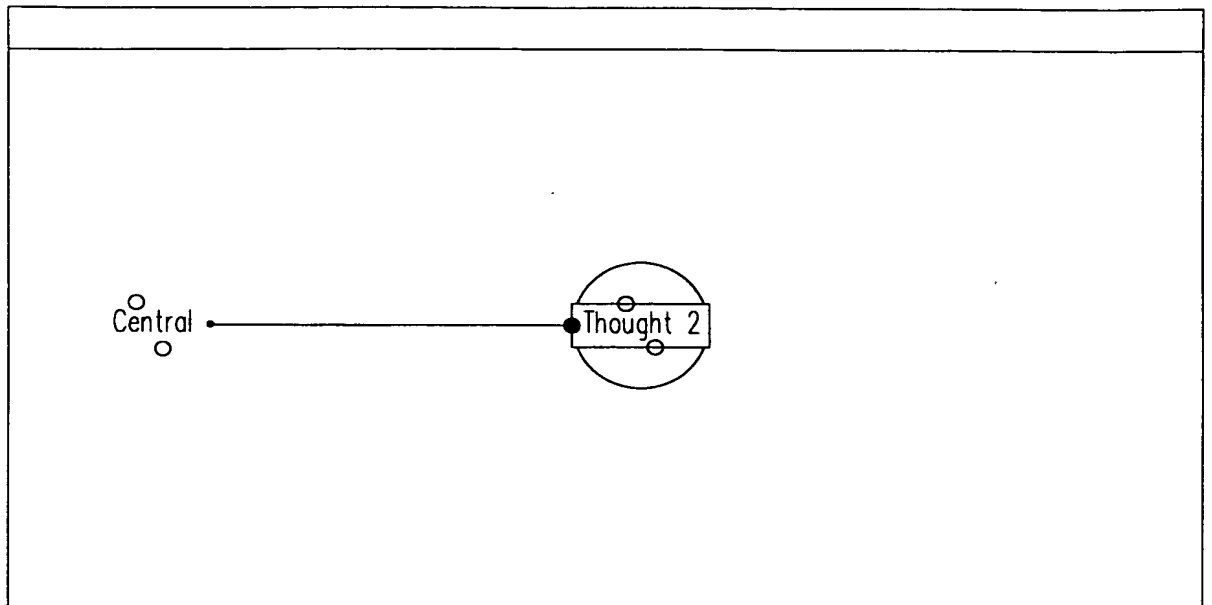


FIG. 19

*FIG. 20*

Spider Site

Name: Lycos
Location: <http://www.lycos.com/>

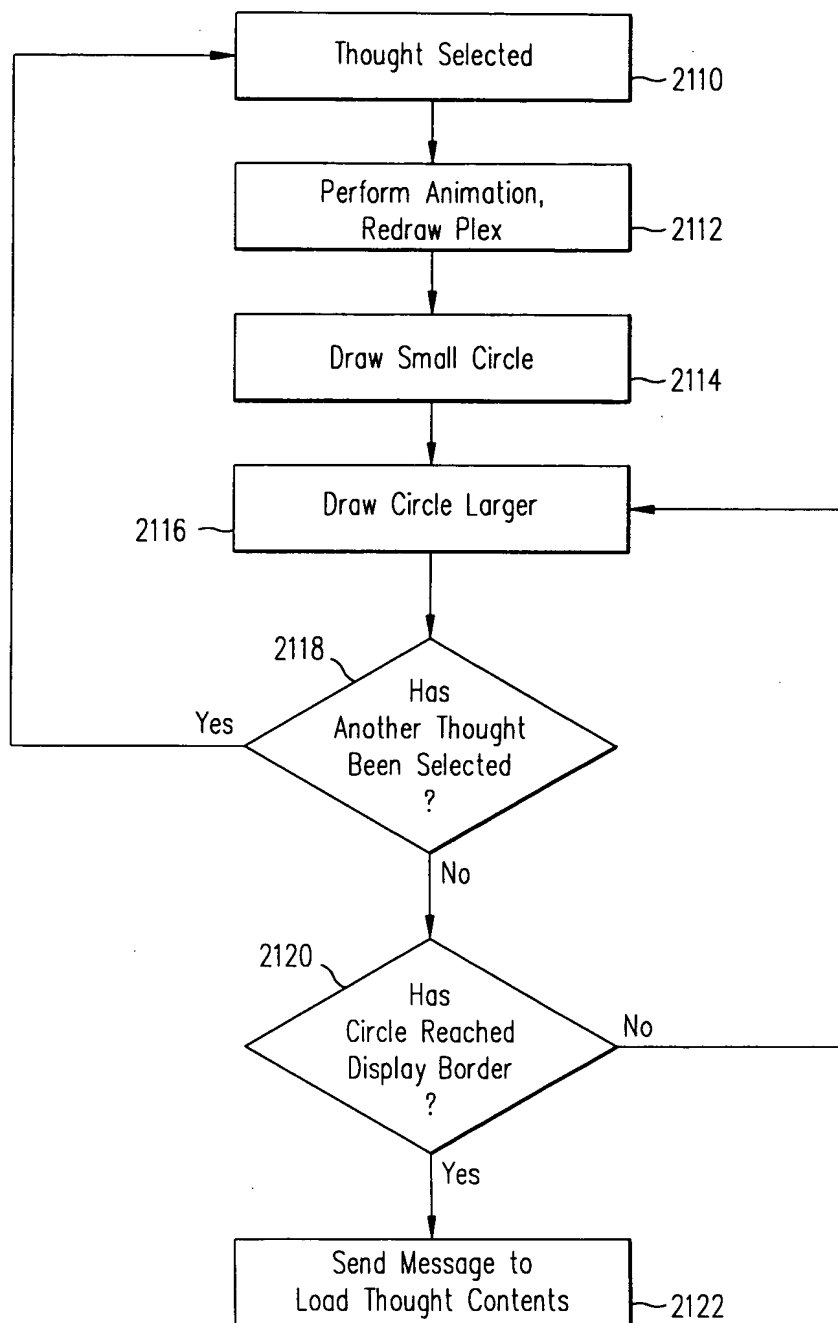
Depth

☐ Add Redundant
☒ Add Non-Local
☐ Spider Non-Local

Done

FIG. 35

FIG. 21

**FIG. 22**

Algorithm for drawing the plex with distant thoughts

1. Create a list of thoughts to be drawn and their on screen locations:
 2. Add the central thought to the list.
 3. Add children to the list.
 4. Add parents to the list.
 5. Add jumps to the list.
 6. Add siblings to the list, checking first that they are not already on the list.
 7. Add distant of children to the list, checking first that they are not already on the list.
 8. Add distant of parents to the list, checking first that they are not already on the list.
 9. Add distant of jumps to the list, checking first that they are not already on the list.
 10. Add distant of siblings to the list, checking first that they are not already on the list.
11. Draw the lines that connect each thought:
 12. For each item in the list:
 13. Get each item in the list:
 14. If the two items are related, draw lines between them from and to the appropriate gates.
15. Draw the distant thoughts:
 16. For each item in the list:
 17. If it is a distant thought, draw it.
18. Draw the other thoughts:
 19. For each item in the list:
 20. If it is not a distant thought, draw it.

FIG. 23

line connects the method for searching thoughts
 lines to find a route from nSrc to nDest other than a direct relation
 returns TRUE if found
 boolean Search(int nSrc, int nDest)

```

{ //create the lists
  ThoughtList posList;    list of thoughts that possibly connect
  ThoughtList notList;    list of thought that do not connect
  //empty the lists
  posList.Initialize();
  notList.Initialize();
  //add the source to the not list since we cannot go directly to the destination
  notList.Add(nSrc);
  //since we cannot go directly to the destination,
  //add all relates except the destination to the possible list
  Thought src(nSrc);
  for(int n = 0; ; n++)
  {
    int nRel = src.GetRelate(n);
    if(!nRel)
    {
      //no more relations, done
      break;
    }
    if(nRel != nDest)
    {
      // add it to the possibly connect list
      posList.Add(nRel);
    }
  }
}

while(TRUE)
{
  //check the first possibility
  int nTest = posList.GetFirst();
  if(!nTest)
  {
    //nothing on the list, done
    break;
  }
  Thought test(nTest);
  if(test.IsRelated(nDest))
  {
    //this one is related to the destination, we're done
    return TRUE;
  }
  //does not connect, add it to the does not connect list
  notList.Add(nTest);
  //add all related thoughts except those already checked to possible list
  for(int n = 0; ; n++)
  {
    int nRel = test.GetRelate(n);
    if(!nRel)
    {
      //no more relations, done
      break;
    }
    if(!notList.Exists(nRel))
    {
      //not checked yet, add to possible list
      posList.Add(nRel);
    }
    //remove this one from the possible list
    posList.Remove(nTest);
  }
  // we've checked everything there is
  // no other way to get from nSrc to nDest
  return FALSE;
}

```

FIG. 24

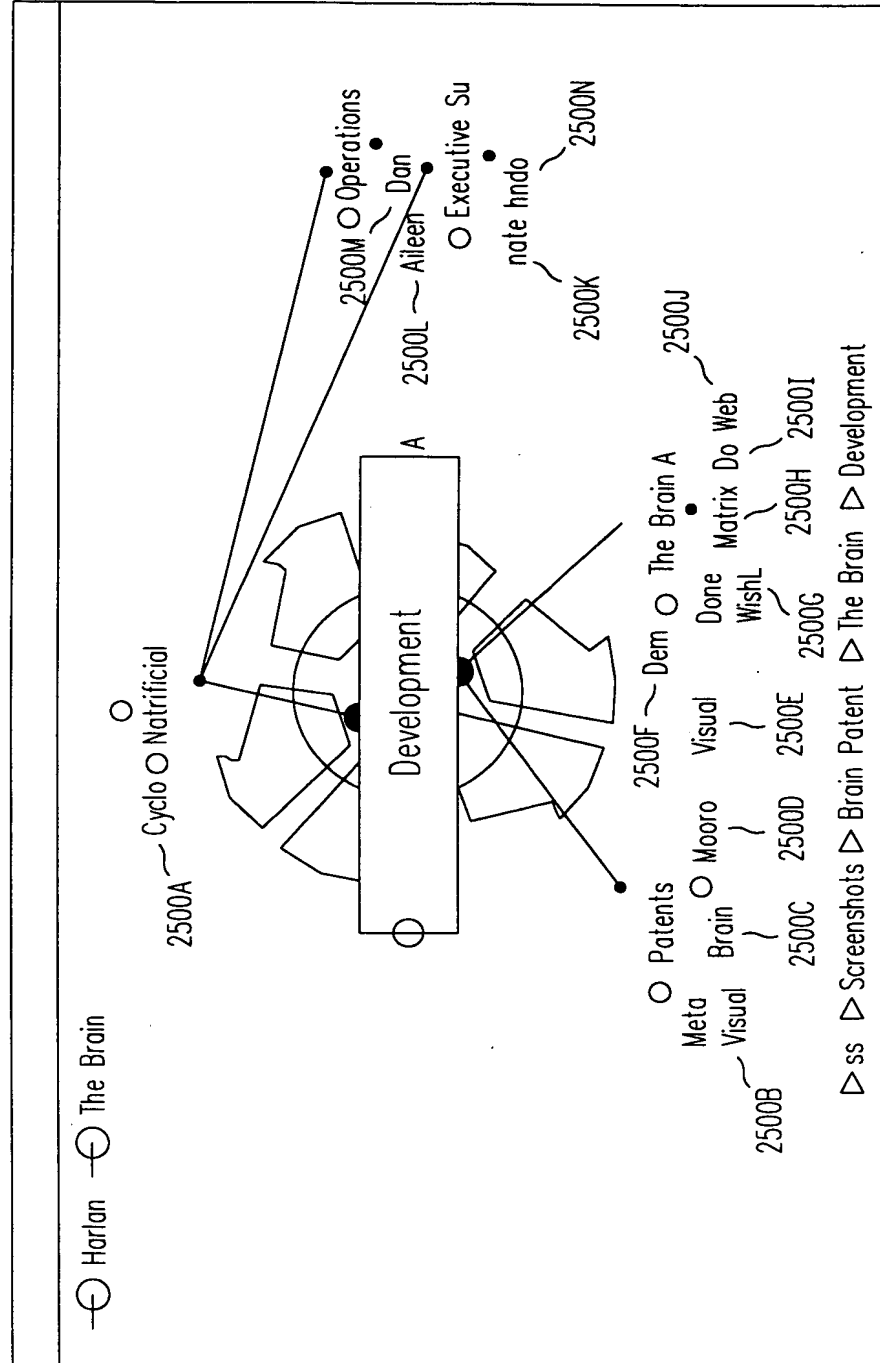


FIG. 25

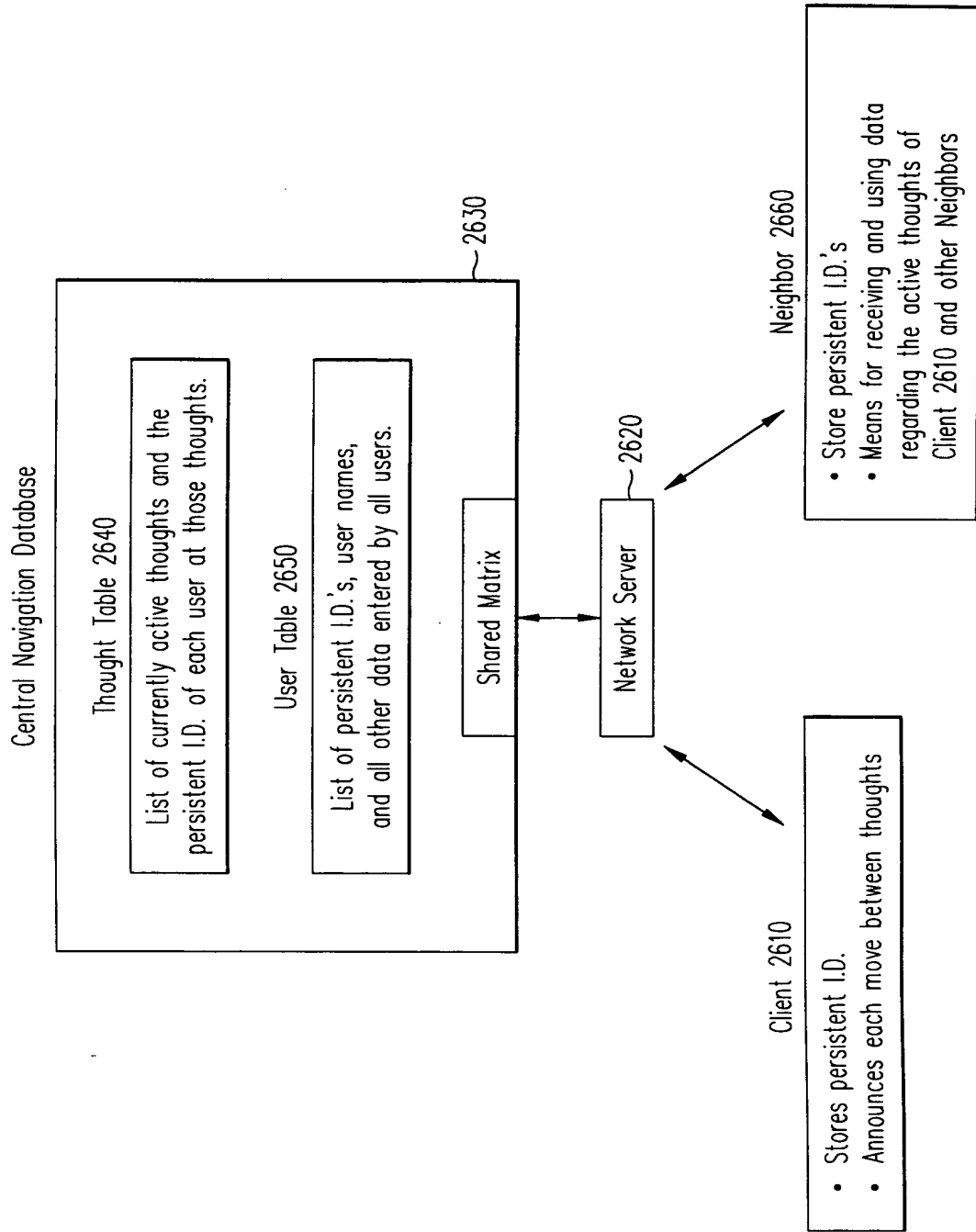


FIG. 26

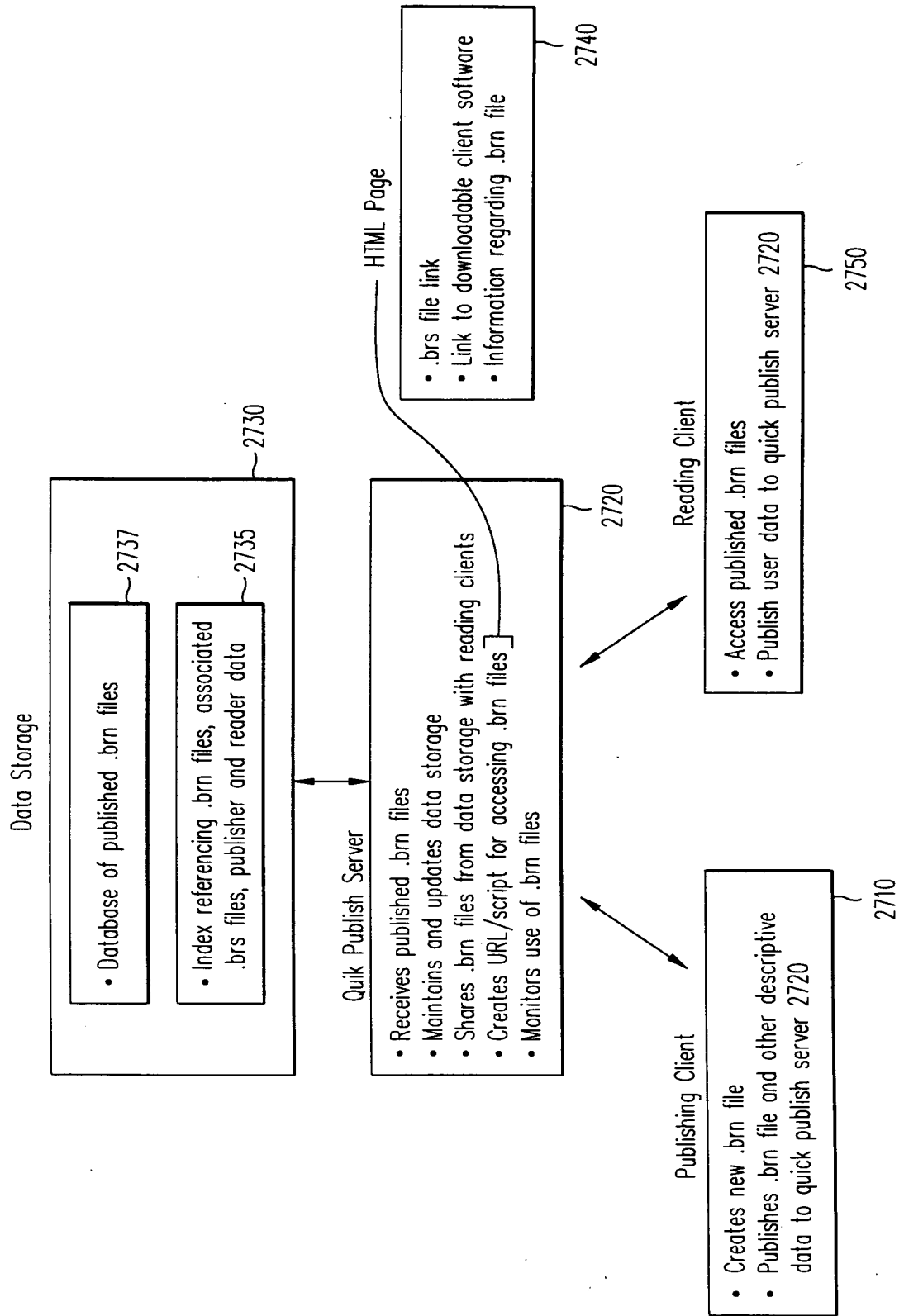


FIG. 27

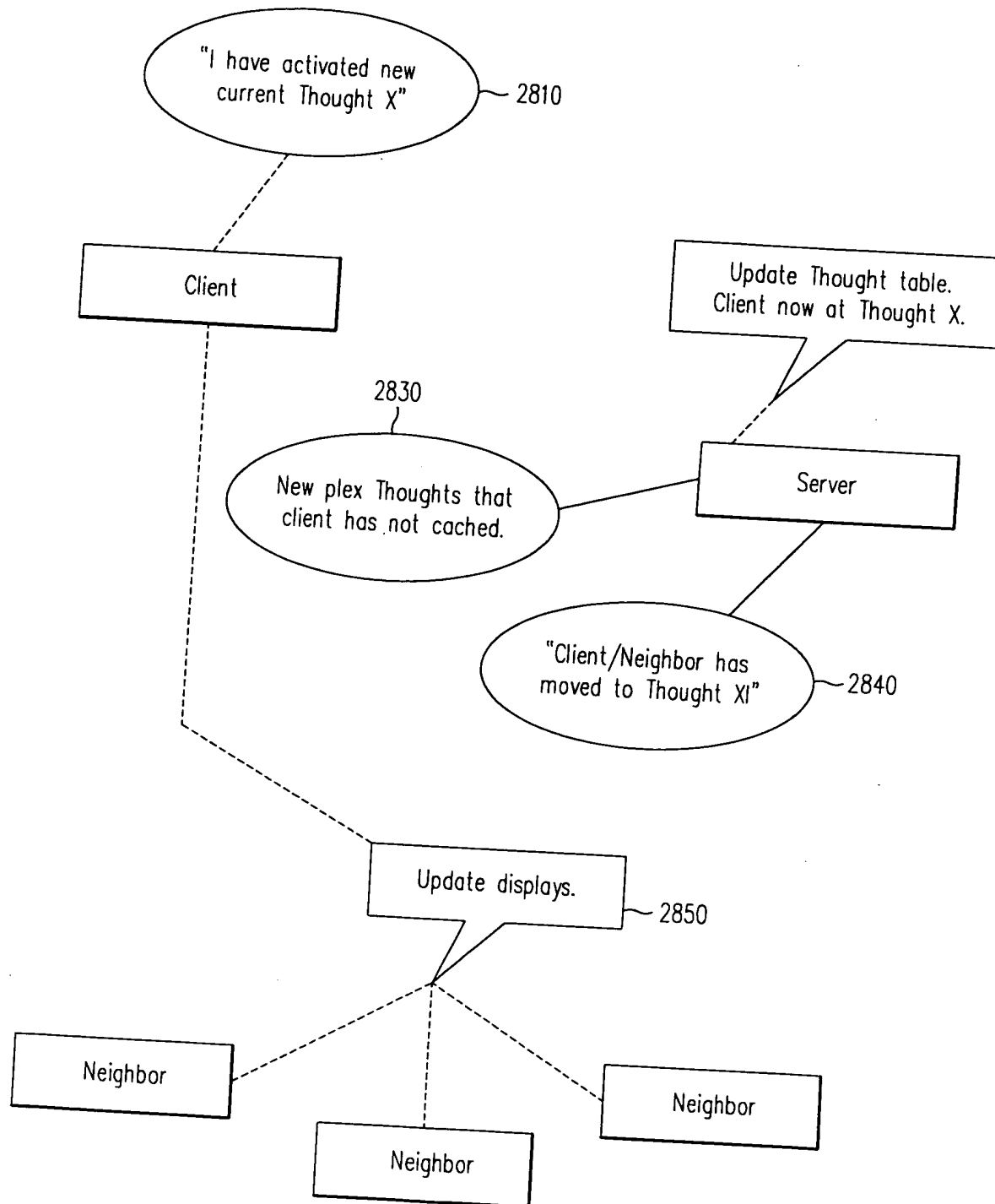


FIG. 28

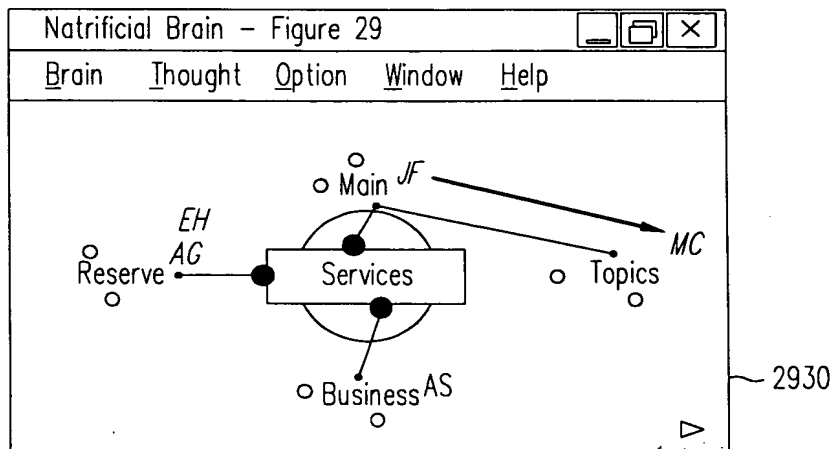
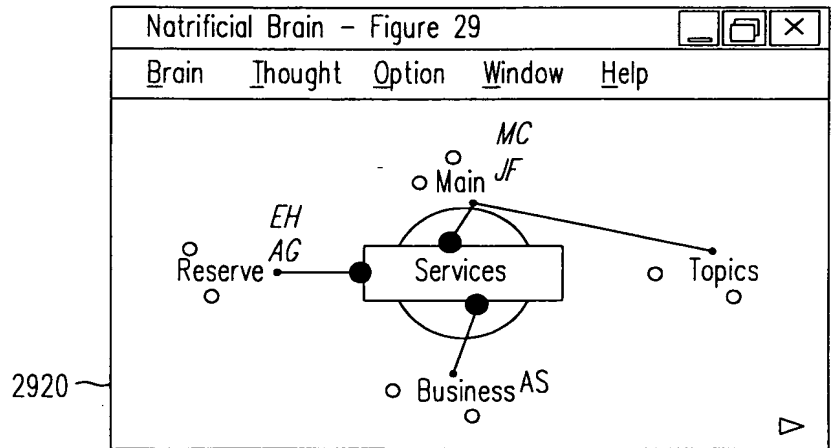
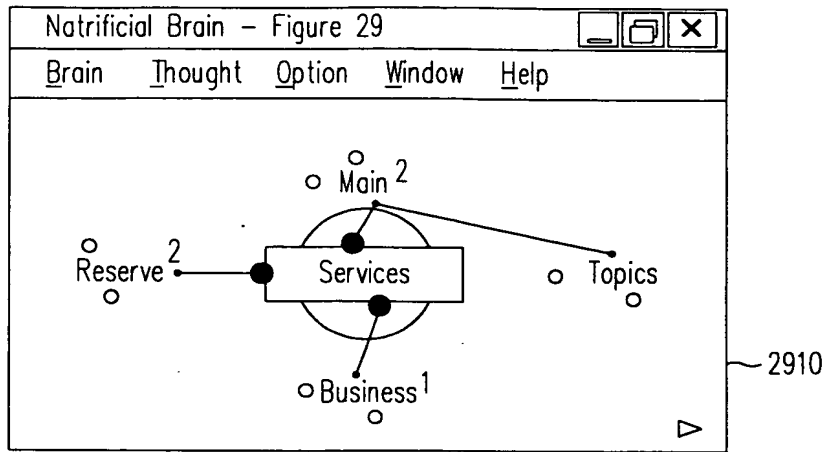


FIG. 29

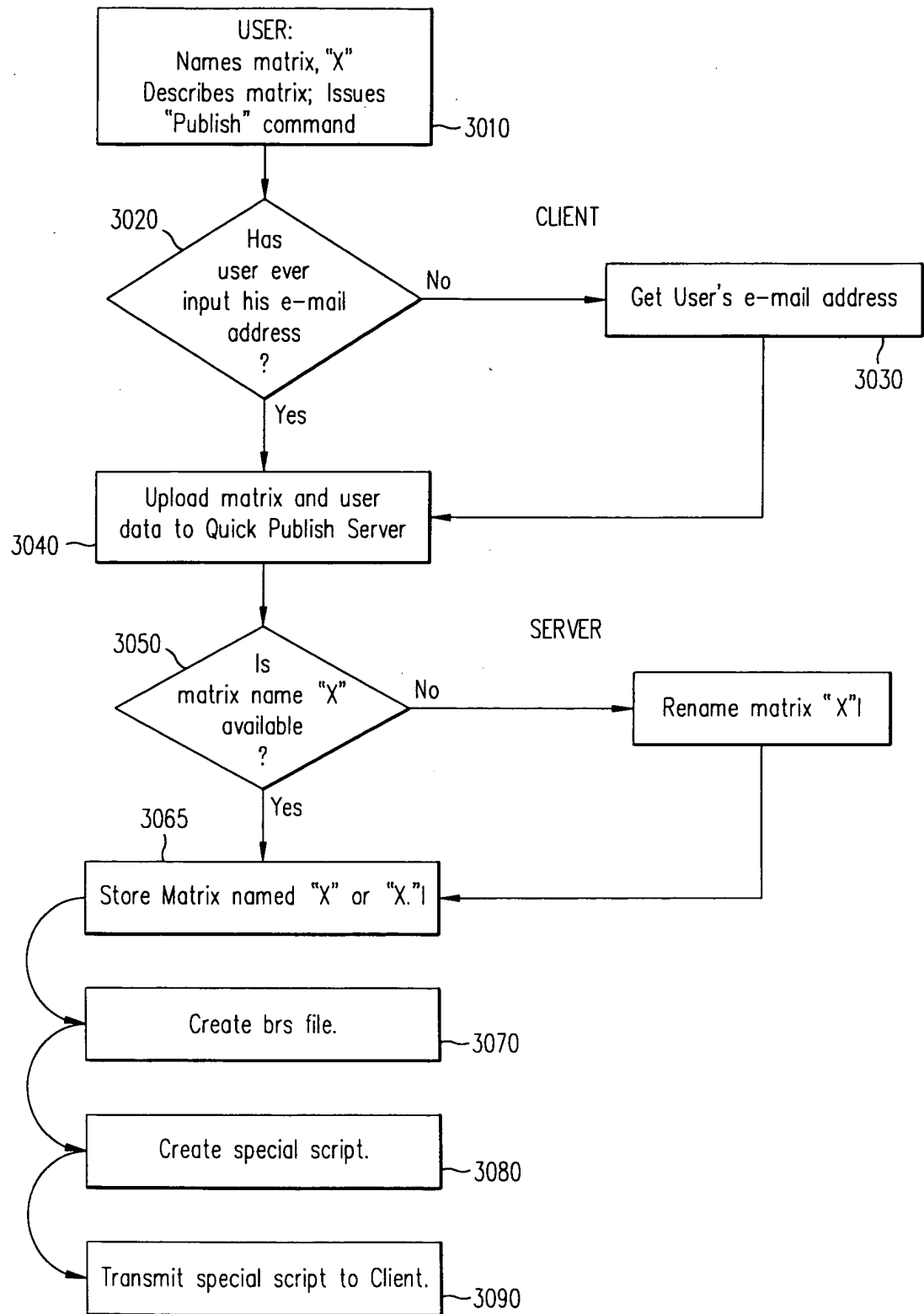


FIG. 30

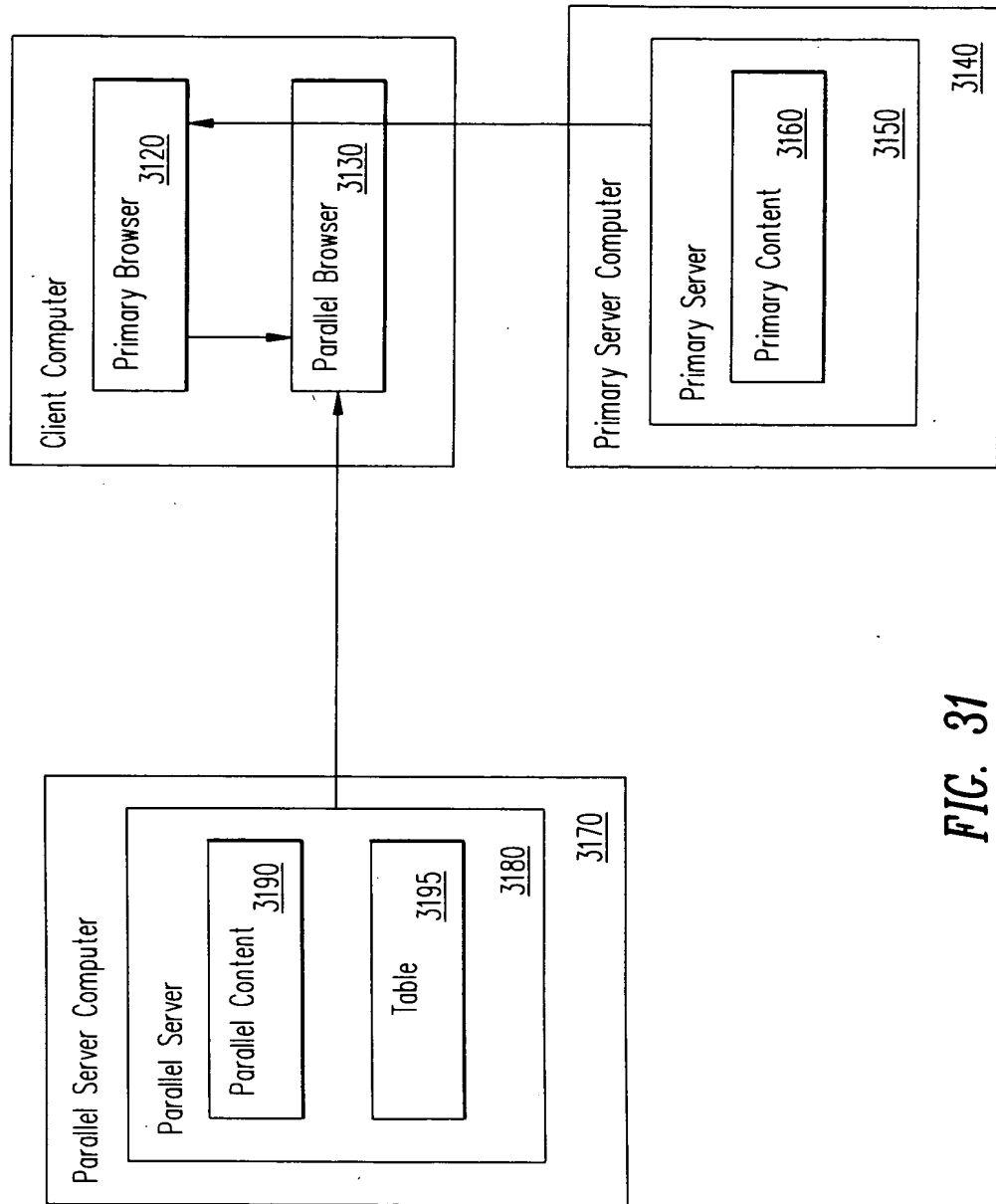
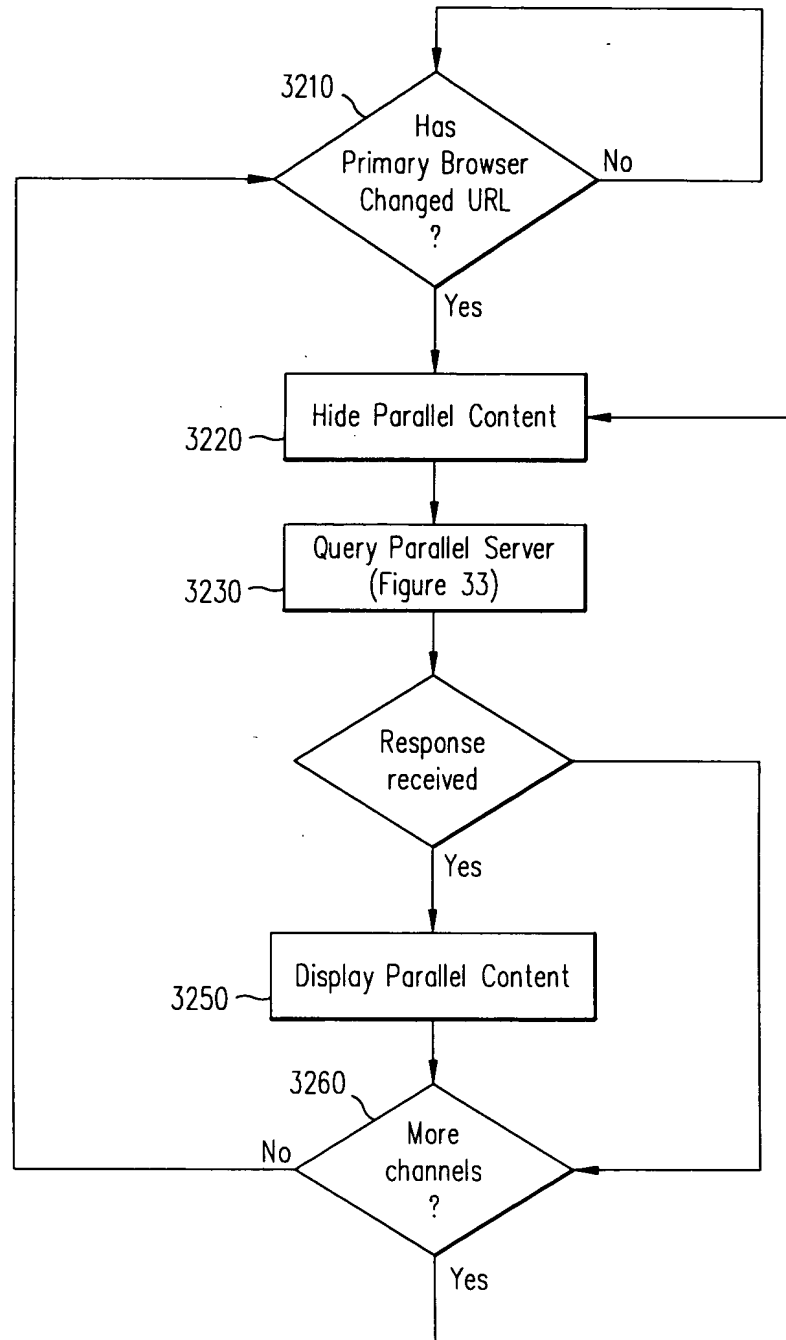


FIG. 31

**FIG. 32**

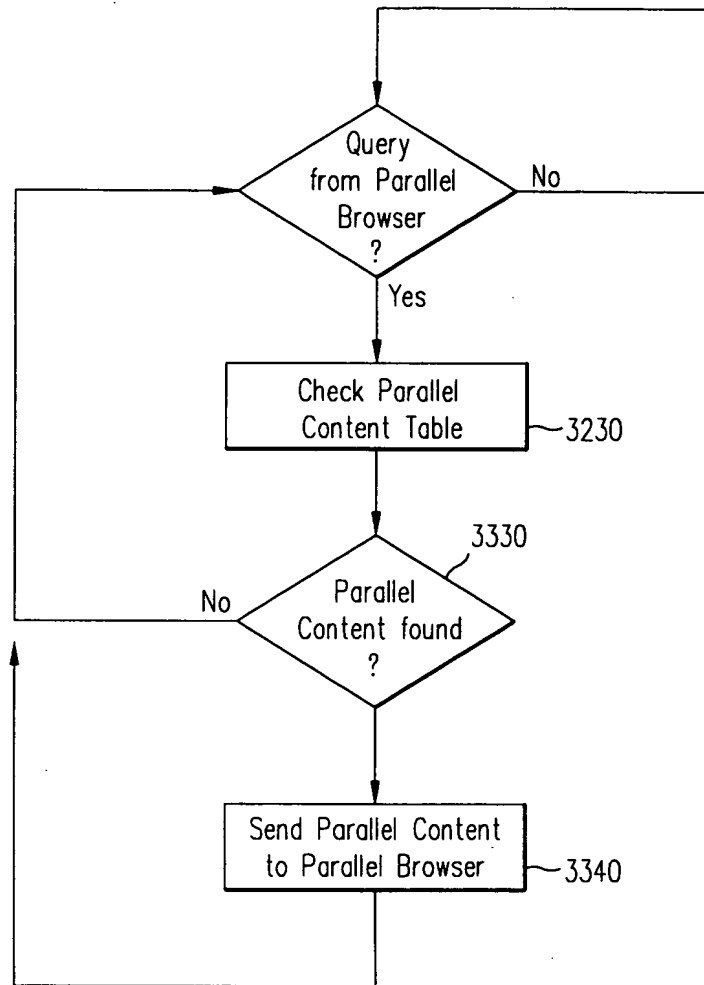
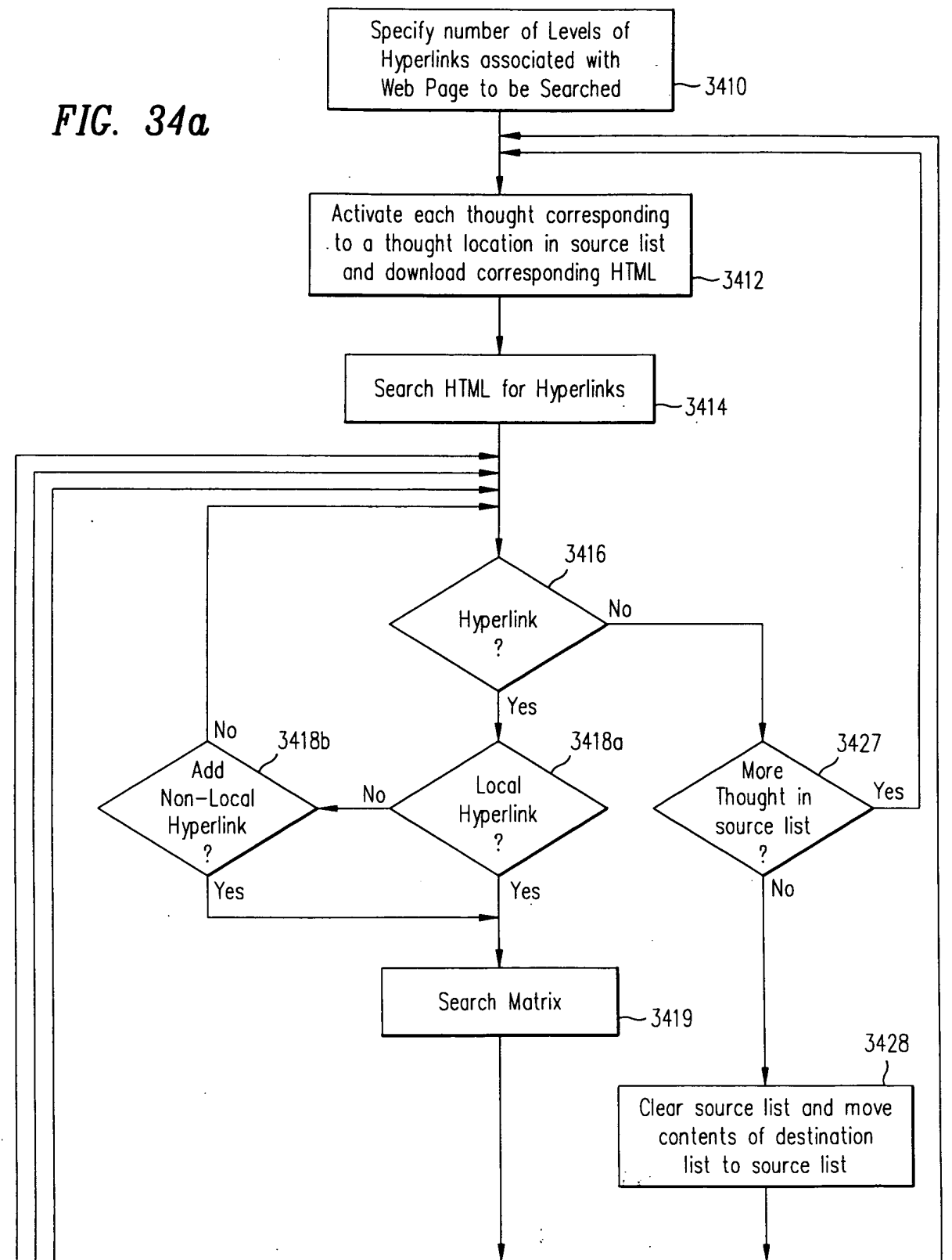
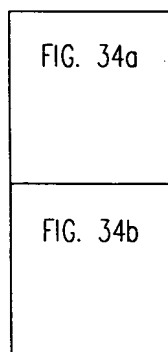
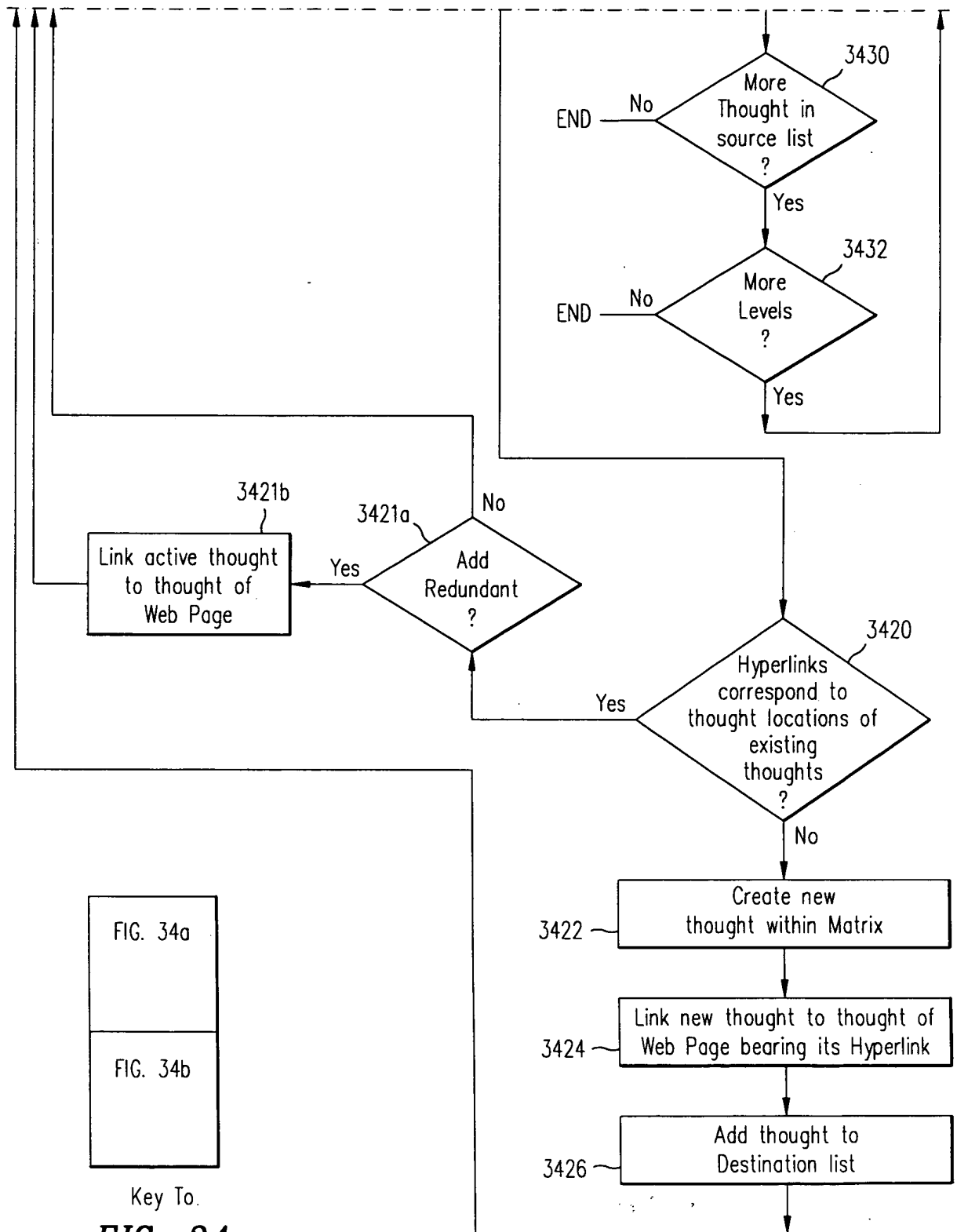
**FIG. 33**

FIG. 34a





Key To.

FIG. 34**FIG. 34b**

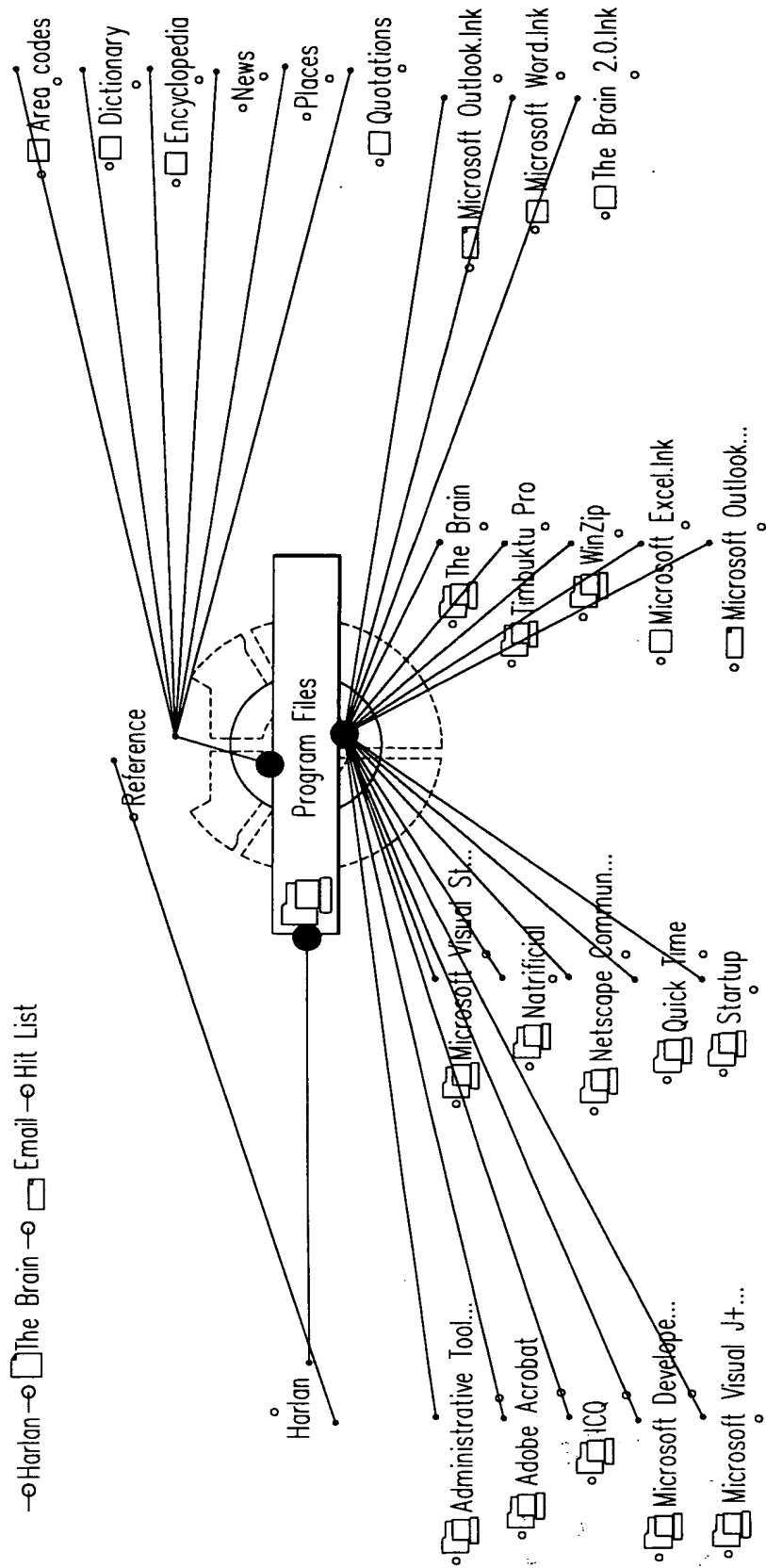
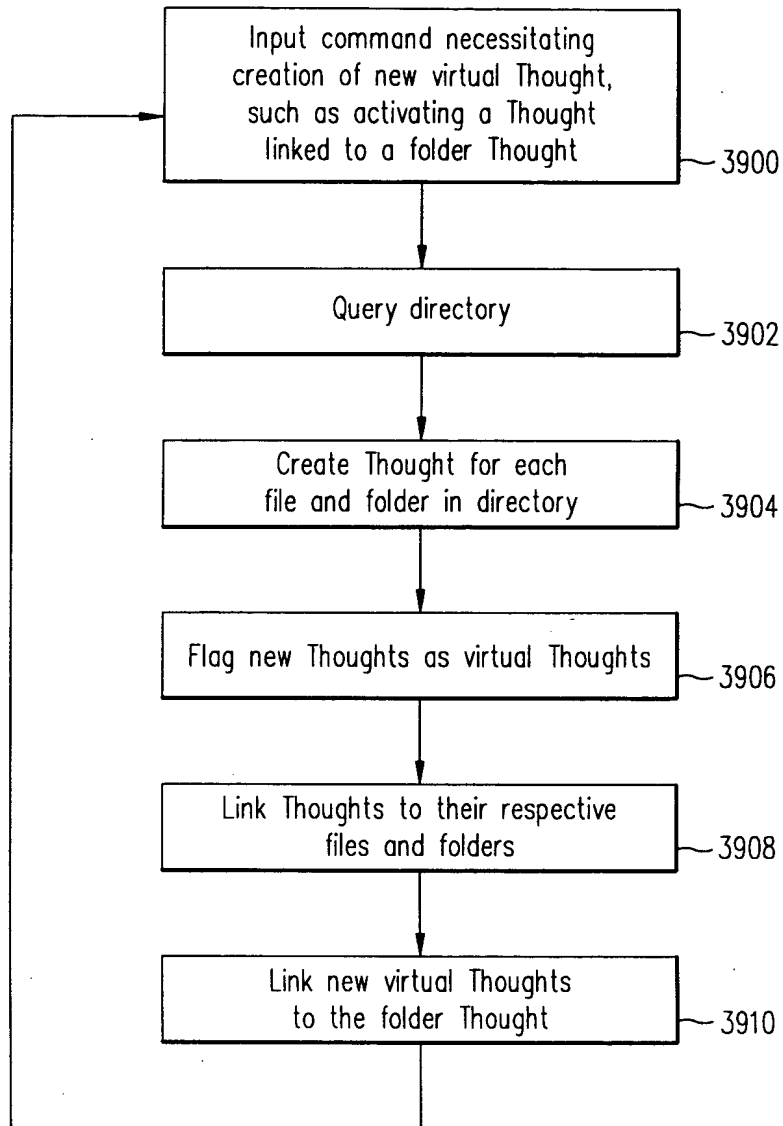


FIG. 36

*FIG. 37*

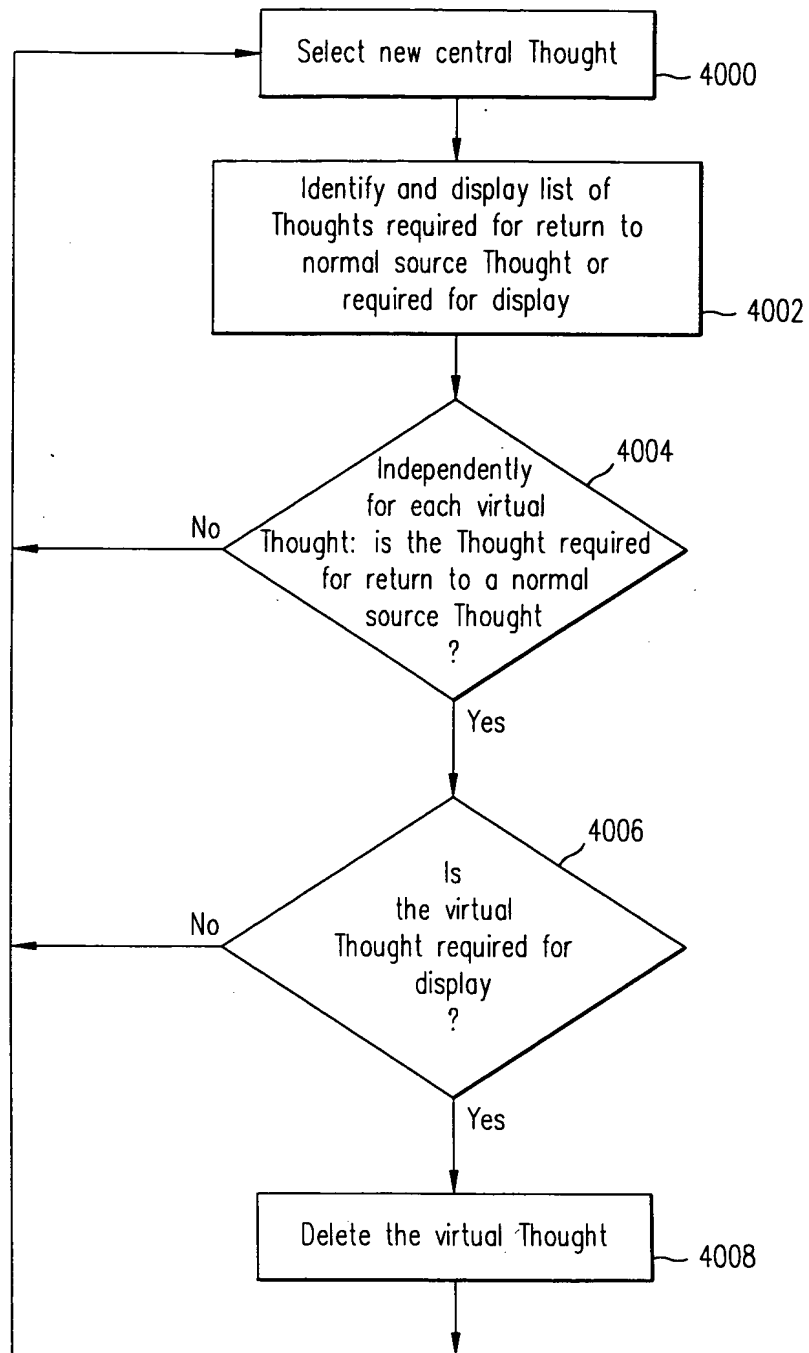


FIG. 38